Κατανεμημένα Συστήματα 1

$1^{η}$ Εργασία

Δούρου Βασιλική Ευαγγελία

A.M.:1072633

Ερώτημα 1:

Ο αλγόριθμος που προτείνεται για να υπολογίσει κάθε διεργασία τον μέσο όρο των αρχικών εισόδων $α_i$ είναι μία παραλλαγή του LCR. Για αυτό θα υποθέσουμε ότι οι διεργασίες επικοινωνούν δεξιόστροφα, αφού δεν διευκρινίζεται η κατεύθυνση και γνωρίζουμε ότι ο δακτύλιος είναι κατευθυνόμενος.

Αρχικά, θα υποθέσουμε ότι κάθε διεργασία i έχει τις ακόλουθες παραμέτρους:

1. Τη μεταβλητή leader, η οποία παίρνει τιμή true ή false, που δηλώνει αν μία διεργασία είναι αρχηγός ή όχι και αρχικά είναι ίση με false.
2. Τη μεταβλητή found, η οποία θα παίρνει τιμή 0 ή 1 και θα δηλώνει αν έχει βρεθεί ο αρχηγός. Αρχικά και αυτή θα είναι ίση με 0.
3. Τη μεταβλητή n, η οποία θα υπολογίσει τον αριθμό των διεργασιών και στην αρχή είναι ίση με 1.
4. Τη μεταβλητή $α_i$, η οποία θα είναι η αρχική είσοδος της κάθε διεργασίας.

5. Τη μεταβλητή u, που θα είναι ίση με το UID της κάθε διεργασίας.
6. Τη μεταβλητή $\alpha_{ολικό}$, που θα χρησιμοποιηθεί για να υπολογίσει το $\sum_i \alpha_i$, με αρχική τιμή το α_i για την κάθε διεργασία.
7. Τη μεταβλητή send, που θα περιέχει την τιμή που θα στείλει η διεργασία i στη διεργασία i+1 και αρχικά θα είναι ίση με τη UID.
8. Τη μεταβλητή $\alpha_{προσωρινό}$ ή $\alpha_{πρ}$, που θα αποθηκεύει την προηγούμενη τιμή που είχε λάβει ως μήνυμα η διεργασία i από τη διεργασία i-1 και θα είναι αρχικά ίση με 0.
9. Τη μεταβλητή $\alpha_{εισ}$ που θα είναι η τιμή που θα έρθει στη διεργασία i από την i-1.

Η γεννήτρια εξερχόμενων μηνυμάτων κάθε διεργασίας i θα είναι της μορφής $msgs_i$: "στείλε το send, το found και το $\alpha_{ολικό}$ στη διεργασία i+1". Με το send θα μπορούμε να εκτελέσουμε κανονικά τον LDR συγκρίνοντας τα UID's των διεργασιών μεταξύ τους, με το found θα ξέρουμε πότε έχει βρεθεί αρχηγός για να στείλει μετά σε όλες τις διεργασίες τον υπολογισμένο μέσο όρο, δηλαδή το $\alpha=\sum_i \alpha_i/n$, και με το $\alpha_{ολικό}$ θα υπολογιστεί το άθροισμα των αρχικών εισόδων.

Αρχικά για όλες τις διεργασίες i θα ισχύει leader=false, found=0, n=1, $\alpha_{πρ}=0$, send=u και $\alpha_{ολικό}=\alpha_i$.

Η συνάρτηση αλλαγής κατάστασης της κάθε διεργασίας i θα είναι της ακόλουθης μορφής $trans_i$, όπου s το UID που ήρθε από τη διεργασία i-1, το f το found που ήρθε από την i-1 και $\alpha_{εισ}$ το $\alpha_{ολικό}$ που ήρθε από την i-1:

if (f=0 and found=0) then

case

s>u: send=s, n=n+1, $\alpha_{ολικό}=\alpha_{ολικό}+\alpha_{εισ}-\alpha_{πρ}$, $\alpha_{πρ}=\alpha_{εισ}$, found=0

s=u: send=null, leader=true, $\alpha_{ολικό}=\alpha_{ολικό}/n$, found=1

s<u: send=null, n=n+1, $\alpha_{ολικό}=\alpha_{ολικό}+\alpha_{εισ}-\alpha_{πρ}$, $\alpha_{πρ}=\alpha_{εισ}$, found=0

endcase

endif

else if (f=1 and found=0) then

send=null, $\alpha_{ολικό}=\alpha_{εισ}$, found=1

endif

else if (f=0 and found=1) then

do nothing

endif

else if (f=1 and found=1) then

do nothing

endif

Σε αυτόν τον αλγόριθμο πρώτα θα εκλεχθεί ο αρχηγός και στη συνέχεια η διεργασία αρχηγός θα υπολογίσει τον συνολικό μέσο όρο και θα προωθήσει τον μέσο όρο στην επόμενη διεργασία. Μετά η επόμενη διεργασία θα προωθήσει τον μέσο όρο στη μεθεπόμενη και ούτω καθεξής, μέχρι κάθε διεργασία στον δακτύλιο να έχει αποθηκευμένη τη σωστή τιμή του μέσου όρου και να ολοκληρωθεί η εκτέλεση του αλγορίθμου.

Ο αλγόριθμος λειτουργεί όπως παρουσιάζεται παρακάτω.

Έστω η διεργασία i. Κάθε διεργασία στέλνει ένα μήνυμα με τρείς μεταβλητές όπως ορίσαμε παραπάνω. Αν στην i έρθει ένα μήνυμα από την i-1, υπάρχουν οι 4 ακόλουθες περιπτώσεις:

1. Δεν έχει εκλεγεί ακόμη αρχηγός, οπότε f=0 και found=0, αφού το found γίνεται 1 μόνο όταν έχει εκλεγεί αρχηγός και το f έρχεται από i-1. Τότε θα συγκρίνει το UID της με το UID που προωθεί η προηγούμενη, δηλαδή με το s. Αν το UID της είναι μικρότερο από αυτό, τότε θα προωθήσει και εκείνη το s, αν είναι ίσο θα εκλεγεί αρχηγός και αν είναι μεγαλύτερο θα συνεχίσει να προωθεί το UID της, όπως λειτουργεί και ο LDR. Η παραλλαγή όμως είναι η εξής, αν δεν έχει εκλεγεί η i αρχηγός, δηλαδή αν το s είναι μεγαλύτερο ή μικρότερο του UID της, τότε θα προσθέσει στο $\alpha_{ολικό}$ της την τιμή $\alpha_{εισ}$, δηλαδή το $\alpha_{ολικό}$ που έρχεται από την i-1 και θα αυξήσουμε το n, δηλαδή τον μετρητή των διεργασιών κατά 1. Επίσης, θα βάλουμε στην μεταβλητή $\alpha_{πρ}$ την τιμή του $\alpha_{εισ}$ για να μην προσθέτουμε πολλές φορές το ίδιο α_i. Αν έχει εκλεγεί η i αρχηγός, τότε θα υπολογίσει το μέσο όρο και θα θέσει το found=1.
2. Έχει εκλεγεί αρχηγός αλλά η διεργασία i δεν το γνωρίζει ακόμη, δηλαδή f=1 και found=0. Τότε η i θα προωθήσει ότι έχει βρεθεί αρχηγός και θα θέσει και το δικό της found=1, θα αποθηκεύσει τον μέσο όρο στο $\alpha_{ολικό}$ της και θα τον προωθήσει στην επόμενη διεργασία.
3. Έχει εκλεγεί αρχηγός και η διεργασία i το γνωρίζει και της προωθείται πάλι ότι έχει βρεθεί αρχηγός, δηλαδή f=1 και found=1, τότε η διεργασία δεν θα κάνει τίποτα παραπάνω, αφού ούτως ή άλλως έχει ήδη αποθηκευμένο τον συνολικό μέσο όρο.
4. Έχει εκλεγεί αρχηγός και η διεργασία i το γνωρίζει, αλλά η i-1 όχι, δηλαδή f=0 και found=1 (η περίπτωση που η i είναι η αρχηγός αλλά η προηγούμενη της δεν έχει ενημερωθεί ακόμη). Τότε πάλι δεν θα κάνει τίποτα παραπάνω και θα αγνοήσει το μήνυμα που της στέλνει η προηγούμενη διεργασία.

Η διεργασία με τη μεγαλύτερη UID εκλέγεται αρχηγός στο τέλος του γύρου n, όπου n ο αριθμός των διεργασιών, αφού για να της έρθει ως s το δικό της UID πρέπει να έχει κάνει το γύρο του δακτυλίου και να έχει περάσει από όλες τις υπόλοιπες διεργασίες. Έπειτα, για να προωθηθεί ο μέσος όρος σε όλες τις διεργασίες και μετά να σταματήσει η διαδικασία πρέπει πάλι να περάσουν n γύροι γιατί σε κάθε γύρο ο σωστός μέσος όρος προωθείται σε μία μόνο διεργασία, ξεκινώντας από τον αρχηγό. Οπότε η χρονική πολυπλοκότητα είναι n+n=2n και είναι της τάξης του O(n), όπου n ο αριθμός των διεργασιών.

Σχετικά με την πολυπλοκότητα επικοινωνίας, για τους πρώτους n γύρους σε κάθε γύρο αποστέλλονται n μηνύματα μέχρι να εκλεγεί ο αρχηγός, αφού κάθε διεργασία στέλνει ένα μόνο μήνυμα στην επόμενη της και οι διεργασίες είναι n. Έπειτα, αφού εκλεγεί αρχηγός, έστω ότι είναι η διεργασία i, αρχικά στέλνονται n μηνύματα, ένα από κάθε διεργασία, μετά n-1, δηλαδή ένα από κάθε διεργασία εκτός από την i, μετά n-2, δηλαδή ένα από κάθε διεργασία εκτός από τις i και i+1, και ούτω καθεξής, δηλαδή συνολικά από το άθροισμα αριθμητικής προόδου στέλνονται $\frac{n(n+1)}{2}$ μηνύματα. Άρα, η πολυπλοκότητα επικοινωνίας είναι $n * n + \frac{n(n+1)}{2} = n^2 + \frac{n^2+n}{2} = \frac{3n^2+n}{2}$ και είναι της τάξης του $O(n^2)$, όπου n ο αριθμός των διεργασιών.

Ερώτημα 2:

Καθώς θεωρούμε ότι έχουμε ένα μη κατευθυνόμενο συνεκτικό δίκτυο και δεν γνωρίζουμε στοιχεία για αυτό, αλλά κάθε διεργασία γνωρίζει τους γείτονες της, προτείνεται μία παραλλαγή του SynchBFS για τον υπολογισμό των πλευρών του δικτύου από έναν διακεκριμένο κόμβο u_0. Αυτός ο

διακεκριμένος κόμβος u_0 θα είναι η ρίζα του δέντρου που θα παραχθεί από τον SynchBFS.

Όλες οι διεργασίες θα έχουν μία μεταβλητή seen, που αρχικά θα είναι ίση με μηδέν και θα γίνεται ένα όταν λάβουν μήνυμα αναζήτησης, μία μεταβλητή parent, που θα αποθηκεύει το UID της διεργασίας από την οποία έλαβε το αρχικό μήνυμα αναζήτησης, και δύο buffer, εκ των οποίων ο ένας θα ονομάζεται $buffer_\varepsilon$ και θα περιέχει τα UID των διεργασιών που έχουν στείλει μήνυμα αναζήτησης στη συγκεκριμένη διεργασία και ο άλλος θα ονομάζεται $buffer_\pi$ και θα αποθηκεύει τα UID των παιδιών της συγκεκριμένης διεργασίας. Με αυτόν τον τρόπο θα γνωρίζουμε πότε έχουμε λάβει μήνυμα από όλους τους γείτονες. Επίσης, η κάθε διεργασία θα έχει και μία μεταβλητή m, η οποία αρχικά θα είναι ίση με μηδέν, με την οποία θα υπολογίζει τις ακμές που συνδέονται σε αυτή. Για να μην υπολογιστεί όμως μία ακμή δύο φορές (μία για κάθε διεργασία στα άκρα της) θα ακολουθούνται οι παρακάτω κανόνες:

1. Αν η ακμή συνδέει διεργασία γονέα με διεργασία παιδί, τότε θα αυξηθεί το m του παιδιού και του γονέα θα μείνει ίδιο.
2. Αν η ακμή συνδέει δύο διεργασίες που δεν έχουν σχέση γονέα παιδιού θα αυξηθεί το m της διεργασίας που έχει το μεγαλύτερο UID (η επιλογή θα μπορούσε να είναι τυχαία, αλλά επιλέχθηκε να αυξηθεί το m της διεργασίας με το μεγαλύτερο UID για λόγους ομοιομορφίας).

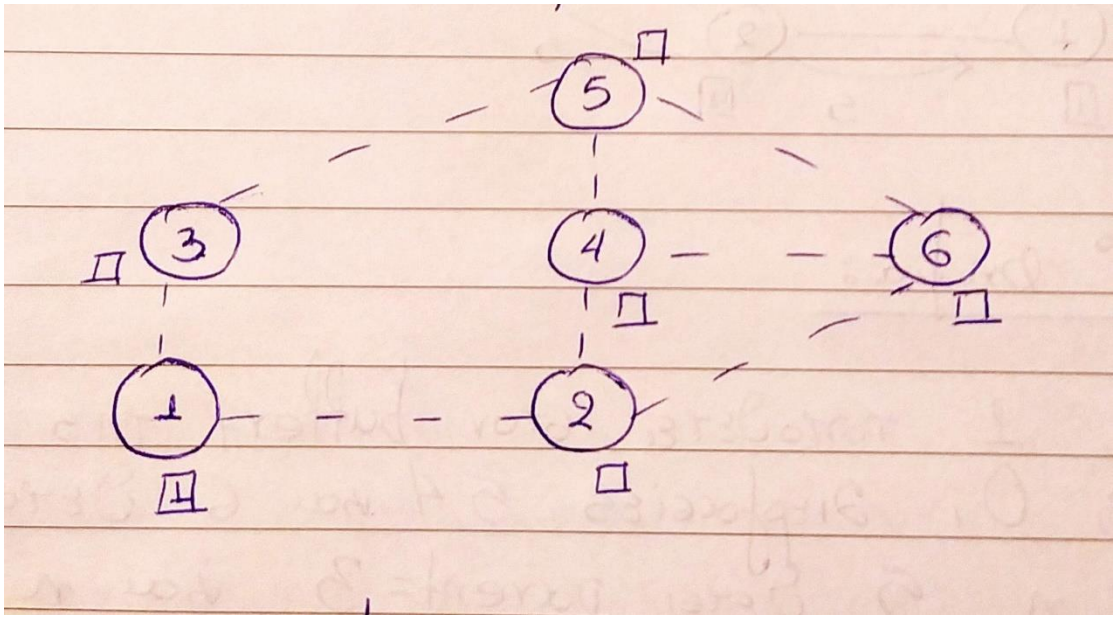
Αν μια διεργασία παρατηρήσει ότι έχει πάρει μήνυμα από όλες τις γειτονικές της διεργασίες, δηλαδή αν είναι όλες στον $buffer_\varepsilon$ της, τότε αν έχει παιδιά, περιμένει τα παιδιά της να της στείλουν τα m τους και αθροίζει το δικό της m και των παιδιών της και στέλνει το άθροισμα στην διεργασία γονέα της, ενώ αν δεν έχει παιδιά, στέλνει το m της στην διεργασία γονέα της.

Αυτό θα συνεχιστεί μέχρι τελικά το ολικό άθροισμα των ακμών να φτάσει στην ρίζα του δέντρου, δηλαδή στην διεργασία u_0 και να τερματίσει ο αλγόριθμος.

Ένα παράδειγμα υλοποίησης του προτεινόμενου αλγορίθμου είναι το ακόλουθο:

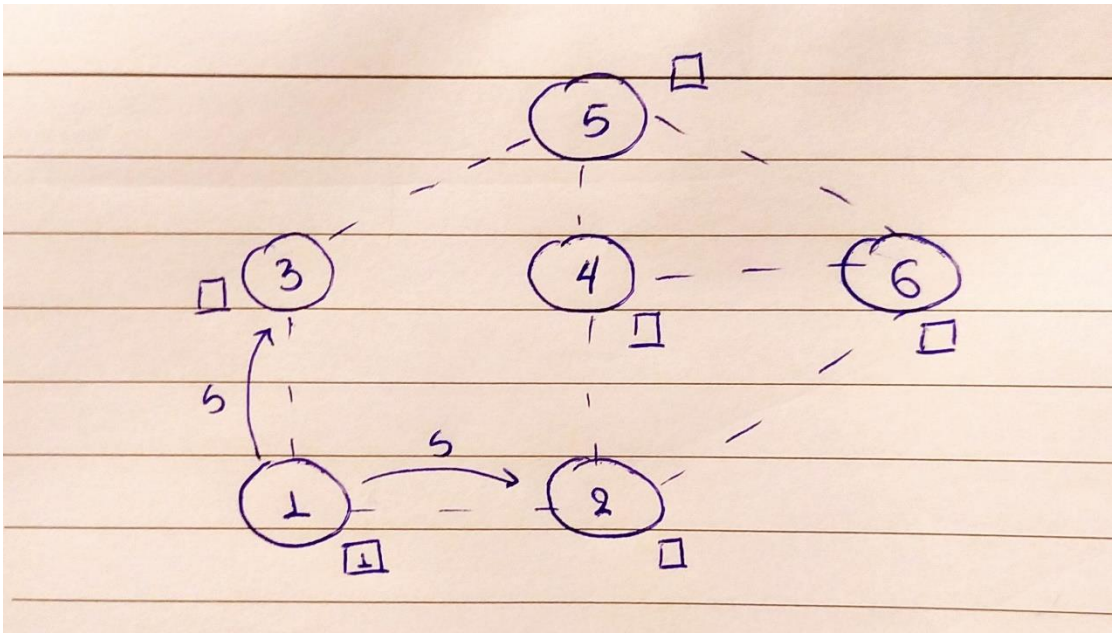
Αρχικό δίκτυο:

Το δίκτυο έχει 6 κόμβους και 8 ακμές. Η διεργασία 1 είναι ο διακεκριμένος κόμβος u_0 και ξεκινά την εκτέλεση του αλγορίθμου. Η διεργασία 1 έχει seen=1 και parent=1, ενώ όλες οι άλλες έχουν seen=0.



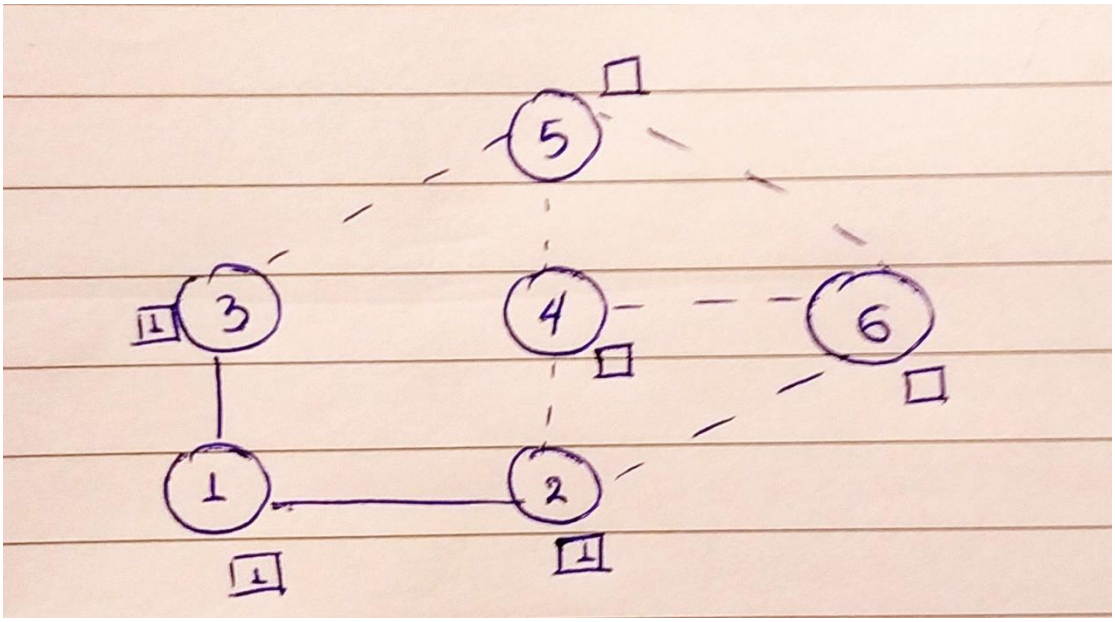
1ος γύρος - 1ο βήμα:

Η διεργασία 1 στέλνει μήνυμα search σε όλους τους γείτονες της.



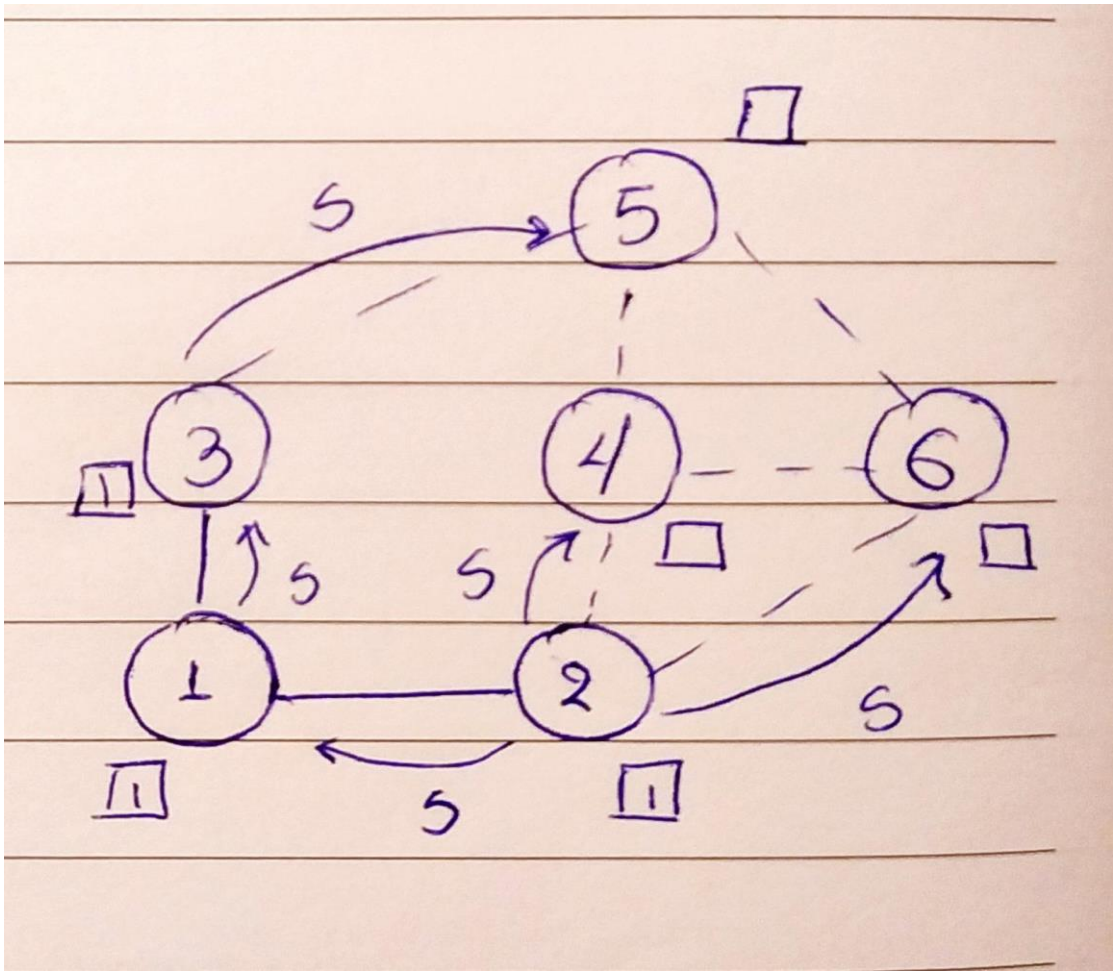
1ος γύρος - 2ο βήμα:

Οι διεργασίες 2 και 3 θέτουν seen=1 και parent=1, ενώ τοποθετούν στον $buffer_\varepsilon$ τους την 1 και αυξάνουν το m τους κατά 1, δηλαδή το θέτουν ίσο με 1.



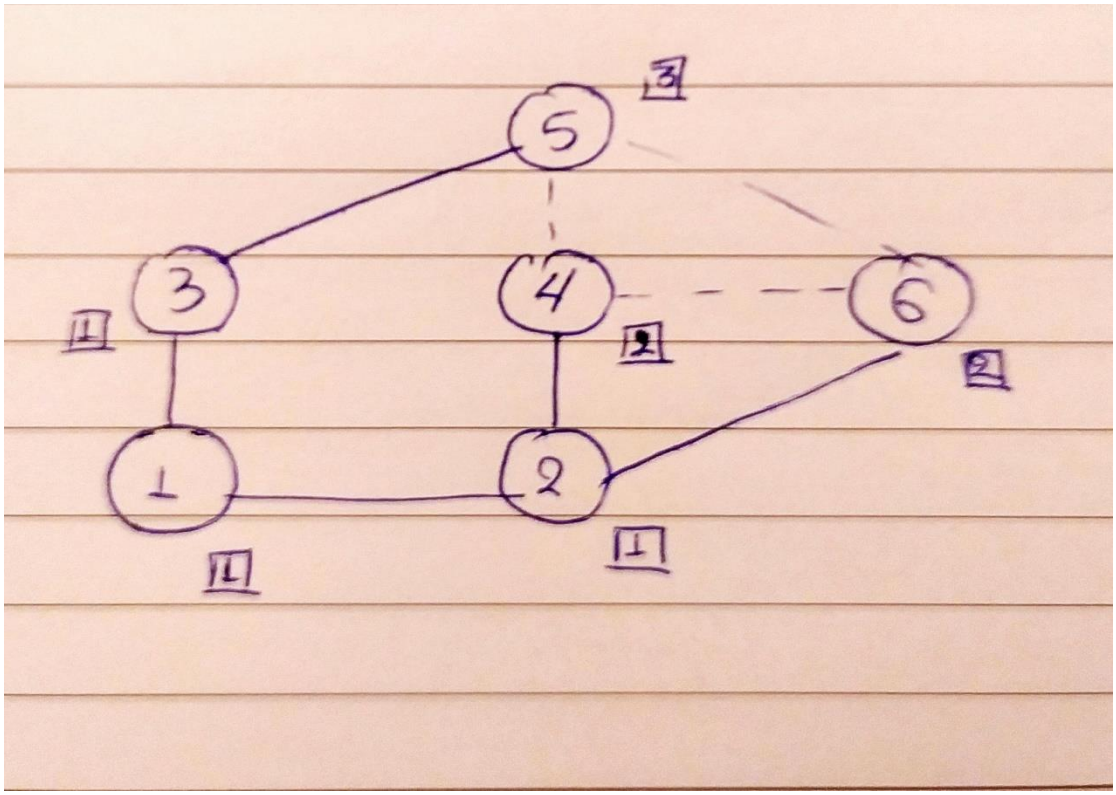
2ος γύρος - 1ο βήμα:

Οι διεργασίες 2 και 3 στέλνουν μήνυμα search σε όλους τους γείτονες τους.



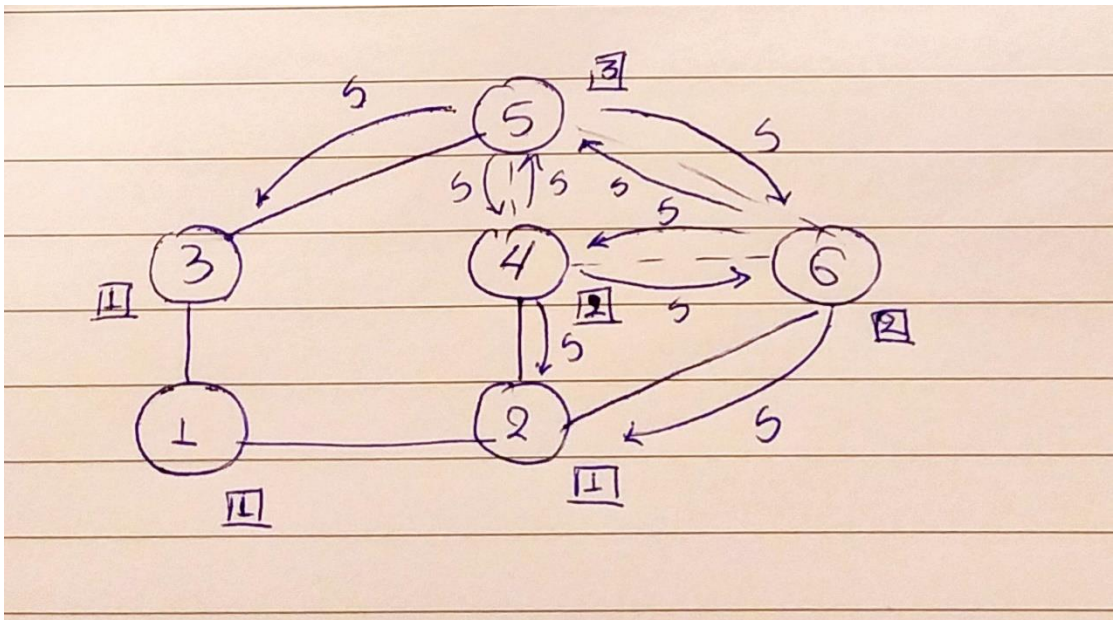
<u>2ος γύρος - 2ο βήμα:</u>

Η διεργασία 1 τοποθετεί στον $buffer_{\pi}$ και στον $buffer_{\varepsilon}$ της τις διεργασίες 2 και 3. Οι διεργασίες 5,4 και 6 θέτουν seen=1, η 5 θέτει parent=3 και οι 4,6 θέτουν parent=2. Επίσης, η 5 τοποθετεί στον $buffer_{\varepsilon}$ της την 3 και οι 4,6 τοποθετούν στους $buffer_{\varepsilon}$ τους την 2. Τέλος, οι 4,5 και 6 θέτουν m=1.



3ος γύρος - 1ο βήμα:

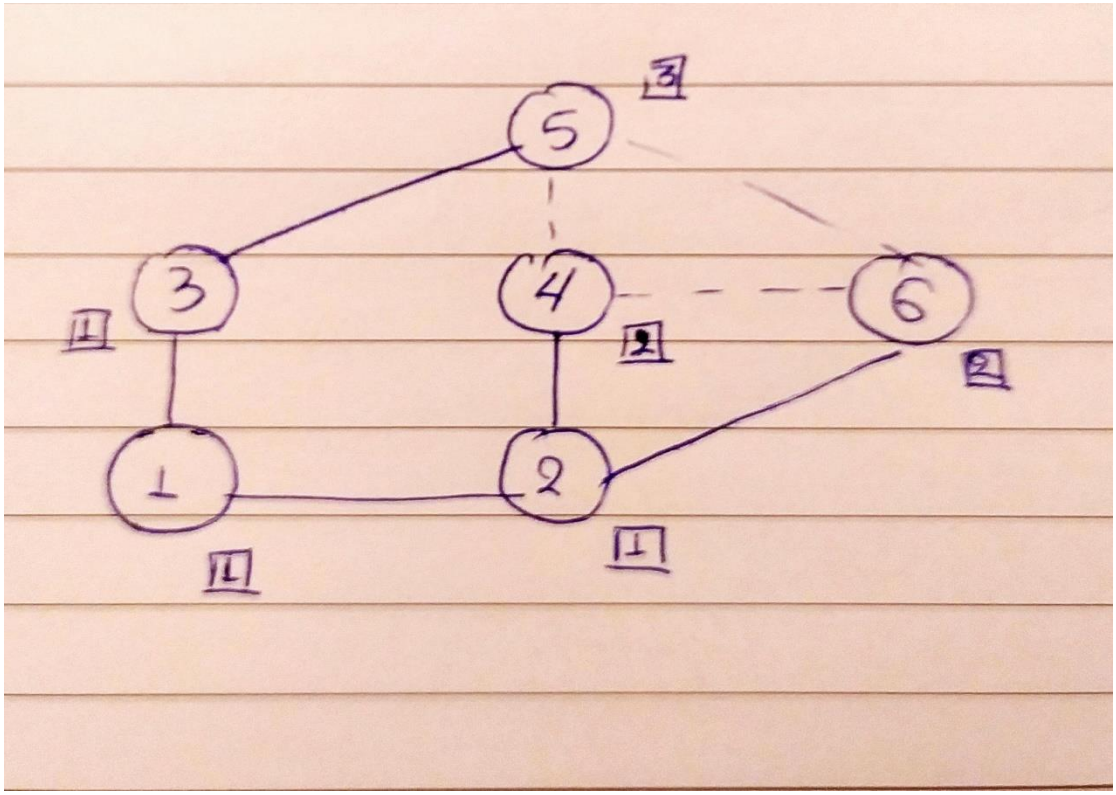
Οι διεργασίες 4,5 και 6 στέλνουν μήνυμα search σε όλους τους γείτονες τους.



3ος γύρος - 2ο βήμα:

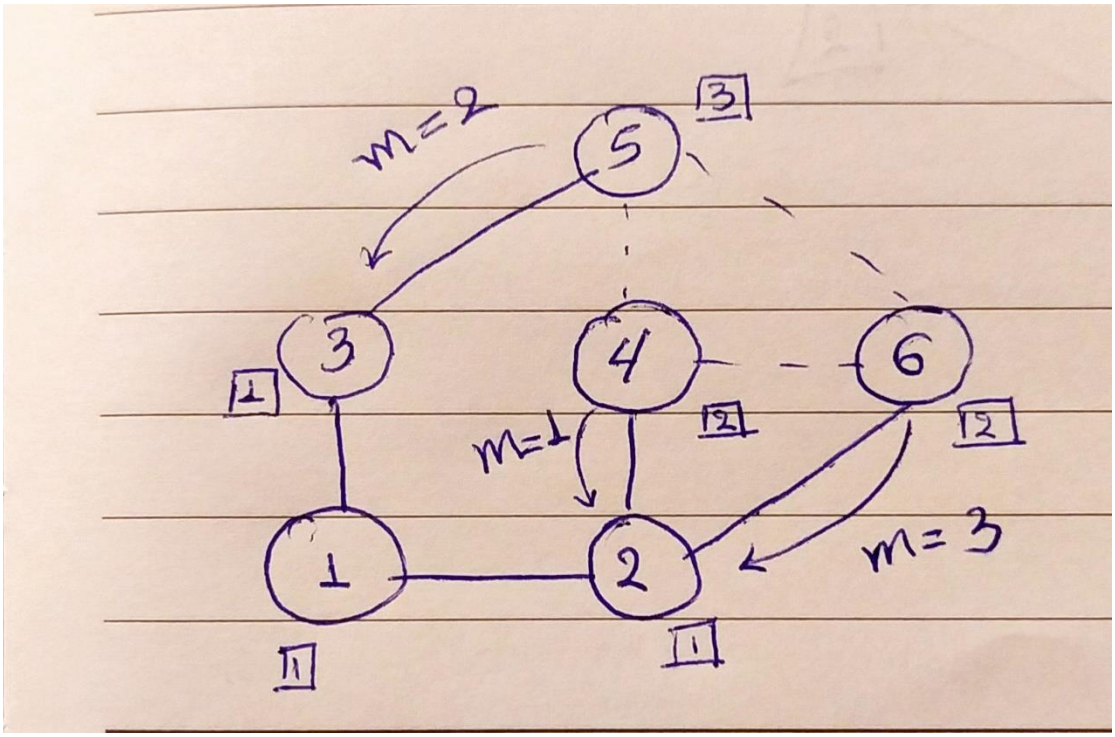
Η διεργασία 2 τοποθετεί στον $buffer_{\pi}$ και στον $buffer_{\varepsilon}$ της τις διεργασίες 4 και 6. Οι διεργασίες 4, 5 και 6 ανταλλάσσουν

μηνύματα search μεταξύ τους. Επειδή όμως 4<5, 4<6 και 5<6, τα μηνύματα μεταξύ της 4 και της 5 θα αυξήσουν μόνο το m της 5, τα μηνύματα μεταξύ της 4 και της 6 θα αυξήσουν μόνο το m της 6 και τα μηνύματα μεταξύ της 5 και της 6 θα αυξήσουν μόνο το m της 6. Άρα τελικά το m της 5 θα γίνει ίσο με 2 και της 6 θα γίνει ίσο με 3. Επίσης, στο $buffer_{\varepsilon}$ της 4 θα προστεθούν οι 5,6, στο $buffer_{\varepsilon}$ της 5 θα προστεθούν οι 4,6 και στο $buffer_{\varepsilon}$ της 6 θα προστεθούν οι 4,5.



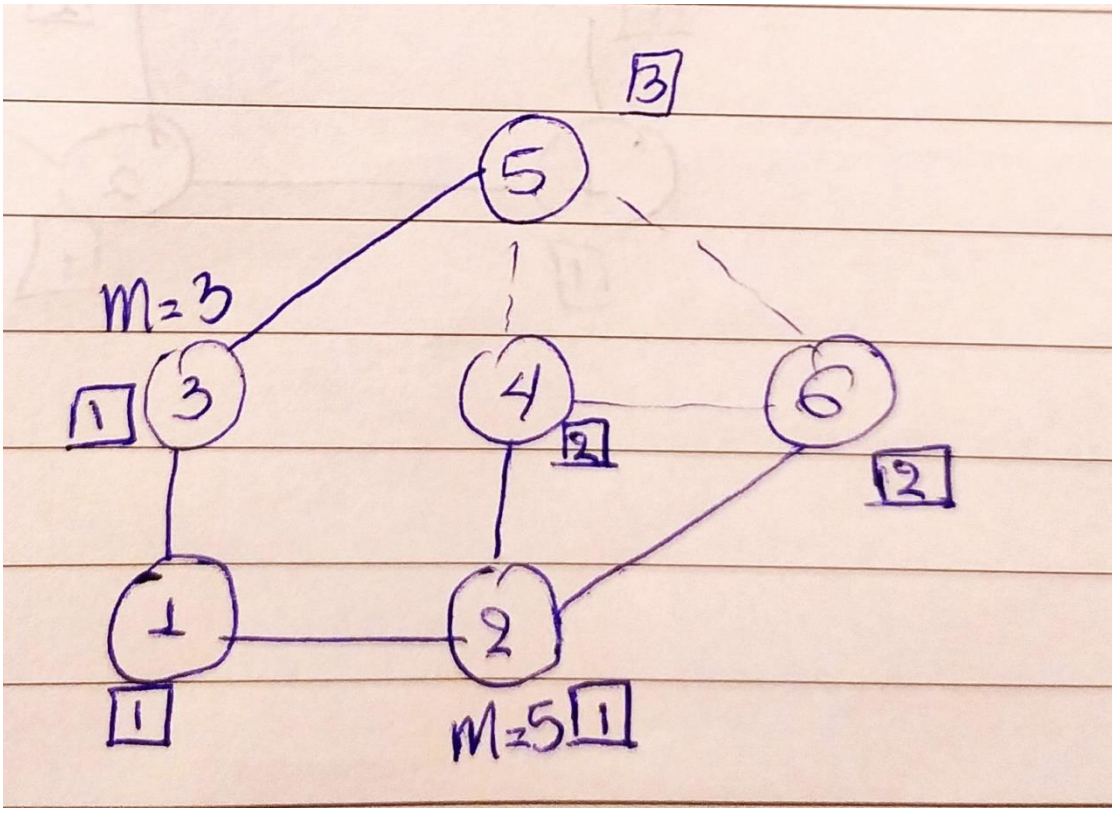
<u>4ος γύρος - 1ο βήμα:</u>

Οι 4,5 και 6 έχουν στους $buffer_{\varepsilon}$ τους όλες τις διεργασίες γείτονες τους και δεν έχουν παιδιά, άρα στέλνουν στους γονείς τους τα m τους.



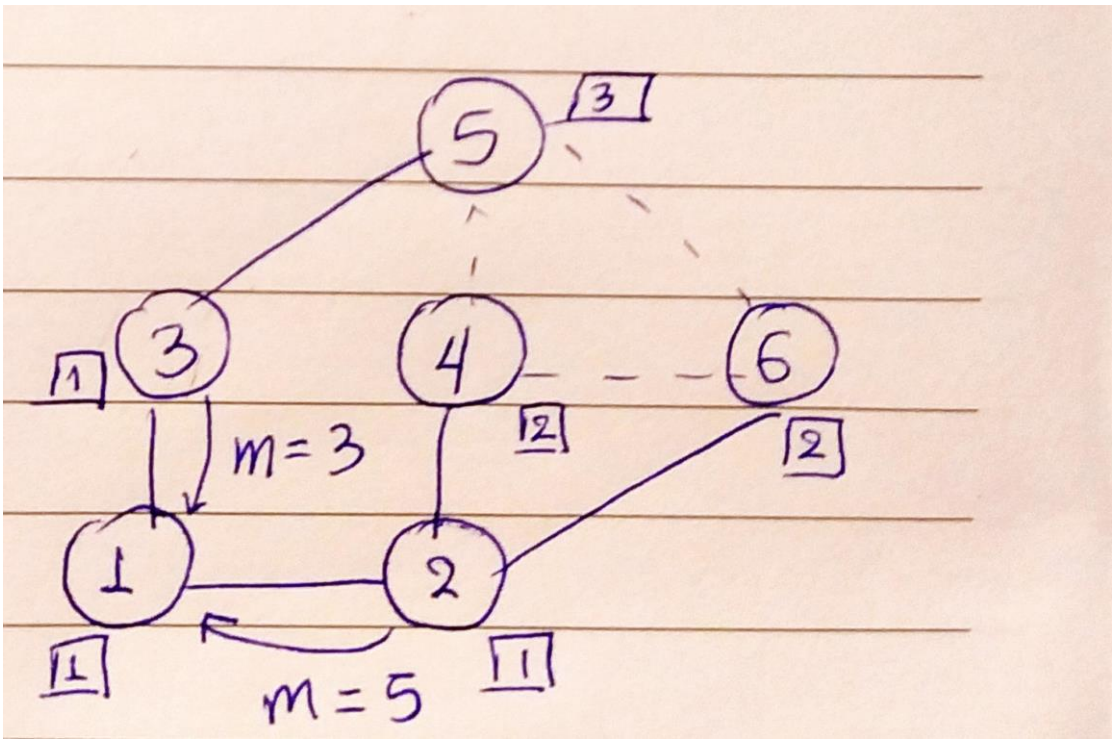
<u>4ος γύρος - 2ο βήμα:</u>

Η 2 και η 3 παραλαμβάνουν τα m των παιδιών τους και άρα υπολογίζουν τα δικά τους m ως 5 (1+1+3) και 3 (1+2) αντίστοιχα.



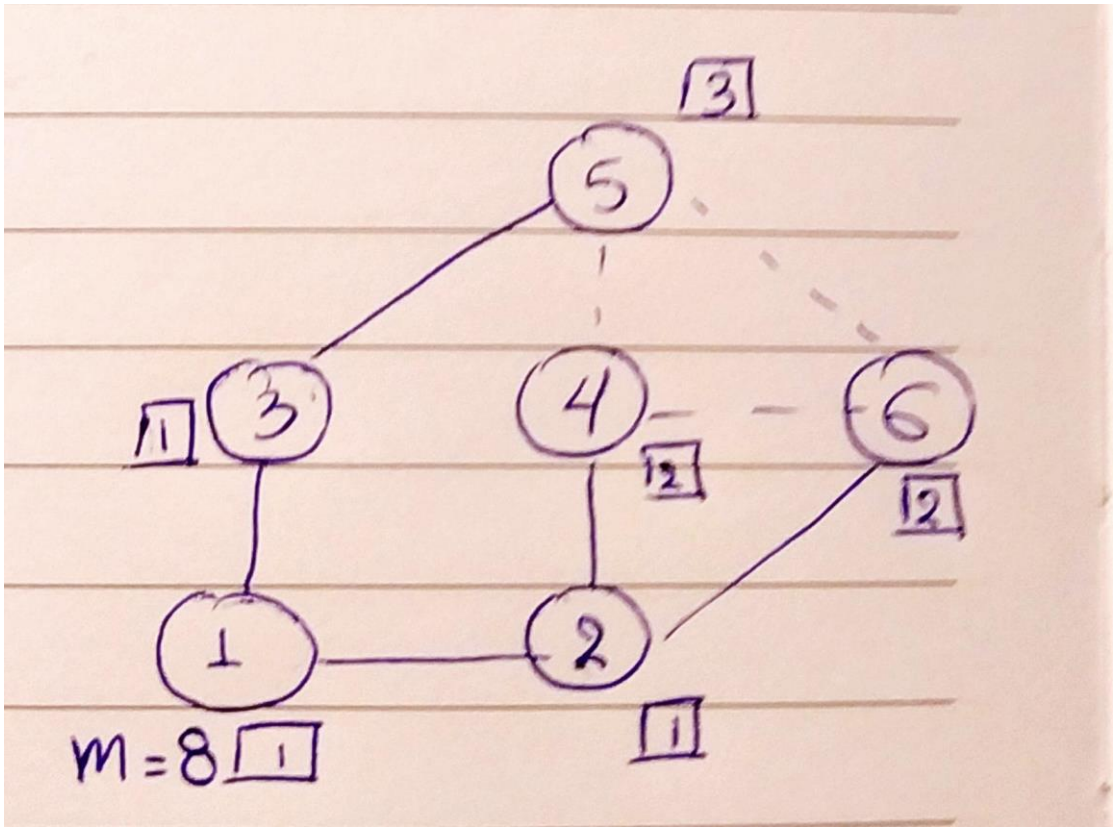
5ος γύρος - 1ο βήμα:

Η 3 και η 2 έχουν στους $buffer_{\varepsilon}$ τους όλες τις διεργασίες γείτονες τους και έχουν λάβει τα m από όλα τα παιδιά τους (τα οποία γνωρίζουν από τους $buffer_{\pi}$). Άρα, στέλνουν στον γονέα τους τα m τους.



5ος γύρος - 2ο βήμα:

Η διεργασία 1 παραλαμβάνει τα m των παιδιών της και υπολογίζει το δικό της m, το οποίο είναι ίσο με 8 (3+5+0).



Η χρονική πολυπλοκότητα του αλγορίθμου είναι ίση με diam+k-1, όπου k ο αριθμός των μη κενών $buffer_{\pi}$. Αυτό ισχύει καθώς αρχικά υλοποιείται ο βασικός SynchBFS με χρονική πολυπλοκότητα ίση με την μέγιστη απόσταση από την u_0, αφού σε κάθε γύρο απομακρυνόμαστε ένα επίπεδο από τη u_0 μέχρι να έχουμε περάσει από όλες τις διεργασίες, και στη συνέχεια ανάλογα με το πόσους ενδιάμεσους κόμβους έχουμε τόσους επιπλέον γύρους χρειαζόμαστε για να φτάσει το σωστό αποτέλεσμα στη ρίζα. Το -1 προστίθεται στο τέλος γιατί και η ρίζα έχει μη κενό $buffer_{\pi}$, αλλά δεν επιβαρύνει χρονικά τον αλγόριθμο γιατί δεν έχει να στείλει κάπου το αποτέλεσμα της για να χρειαστεί και άλλο γύρο.

Η πολυπλοκότητα επικοινωνίας του αλγορίθμου είναι ίση με 2m+n-1, όπου n το σύνολο των κόμβων. Το 2m οφείλεται στο γεγονός ότι ανάμεσα σε 2 διεργασίες που επικοινωνούν στέλνονται δύο μηνύματα search, ένα από την κάθε μία, άρα

σε κάθε ακμή αντιστοιχούν 2 μηνύματα, οπότε συνολικά 2m. Το n-1 οφείλεται στο γεγονός ότι ο διακεκριμένος κόμβος u_0, δηλαδή η ρίζα του δέντρου, πρέπει να δεχτεί όλα τα m που έχουν υπολογίσει τα παιδιά του και τα παιδιά των παιδιών του και ούτω καθεξής, για να μπορέσει να μετρήσει όλες τις ακμές του γραφήματος (το -1 μπαίνει στο τέλος για να αφαιρεθεί η ρίζα, αφού δεν έχει να στείλει κάπου το αποτέλεσμα της και έτσι δεν επιβαρύνει επιπλέον τον αλγόριθμο).